**CENTRO UNIVERSITÁRIO DE SANTA FÉ DO SUL - UNIFUNEC**

**MATHEUS DE SOUZA PEREIRA**

**DOCUMENTAÇÃO TÉCNICA**

**PROJETO MEMOTECA DE RECEITAS**

**SANTA FÉ DO SUL, SP**

**2025**

**1. Introdução e Visão Geral**

O projeto "Memoteca de Receitas" é uma aplicação web desenvolvida para permitir aos usuários catalogar, visualizar, editar e excluir receitas culinárias, com um foco especial em preservar as histórias e memórias associadas a cada prato. A aplicação consiste em um frontend interativo construído com angular (utilizando componentes standalone) e um backend PHP que gerencia a persistência dos dados em um banco de dados MySQL, operando sobre um ambiente XAMPP.

**Funcionalidades Principais:**

* Criação, Leitura, Atualização e Exclusão (CRUD) de receitas.
* Upload e exibição de imagens associadas às receitas.
* Interface de usuário reativa com listagem em cards e visualização detalhada.
* Navegação baseada em rotas.

**2. Arquitetura da Aplicação**

A aplicação segue uma arquitetura cliente-servidor clássica:

* **Frontend (Cliente):** Aplicação Single-Page Application (SPA) desenvolvida em Angular. Responsável pela interface do usuário, interações, validações do lado do cliente e comunicação com o backend via requisições HTTP.
* **Backend (Servidor):** API RESTful desenvolvida em PHP. Responsável pela lógica de negócios, validações do lado do servidor, interações com o banco de dados e manipulação de arquivos (upload de imagens).
* **Banco de Dados:** MySQL, gerenciado através do XAMPP. Armazena os dados das receitas.
* **Servidor Web:** Apache (parte do XAMPP), serve os arquivos da API PHP e, indiretamente, as imagens.

**Fluxo de Comunicação:** O frontend Angular (executando no navegador do usuário) envia requisições HTTP (GET, POST, DELETE) para os scripts PHP da API. A API processa essas requisições, interage com o banco de dados MySQL e retorna respostas (geralmente em formato JSON) para o frontend, que então atualiza a interface do usuário.

**3. Frontend (Angular)**

O frontend foi construído utilizando Angular com foco em componentes standalone, o que simplifica a estrutura do projeto ao eliminar a necessidade de NgModules para cada componente.

**3.1. Estrutura de Pastas Principal (dentro de src/app/)**

* app.component.{ts,html,css}: Componente raiz da aplicação.
* app.config.ts: Configurações da aplicação, incluindo provedores de roteamento e HttpClientModule.
* app.routes.ts: Definição das rotas da aplicação.
* receita.ts: Interface TypeScript definindo a estrutura de dados de uma Receita.
* receita.service.ts: Serviço responsável pela comunicação com a API backend.
* receitas/: Pasta contendo os componentes relacionados às receitas:
  + receita-card/: Componente para exibir um card de receita.
  + receita-detalhe/: Componente para exibir os detalhes de uma receita.
  + receita-form/: Componente do formulário de criação/edição de receita.
  + receita-lista/: Componente para listar todas as receitas.
* shared/: Pasta para componentes compartilhados (ex: header/).

**3.2. Componentes Principais**

* **AppComponent (app.component.ts)**:
  + Componente raiz que hospeda o RouterOutlet para renderização das rotas e o HeaderComponent.
  + Define o layout básico da página (cabeçalho, área de conteúdo principal, rodapé).
* **HeaderComponent (shared/header/header.component.ts)**:
  + Componente reutilizável para o cabeçalho da aplicação.
  + Contém links de navegação (RouterLink) para as principais seções (ex: home, adicionar receita).
* **ReceitaListaComponent (receitas/receita-lista/receita-lista.component.ts)**:
  + Responsável por buscar e exibir a lista de todas as receitas.
  + Utiliza o ReceitaService para obter os dados do backend.
  + Itera sobre a lista de receitas e renderiza um ReceitaCardComponent para cada uma.
  + Manipula a lógica de exclusão de receitas, chamando o serviço e atualizando a lista.
* **ReceitaCardComponent (receitas/receita-card/receita-card.component.ts)**:
  + Componente de apresentação que recebe um objeto Receita como @Input().
  + Exibe as informações resumidas da receita (título, autor, data, imagem).
  + Contém botões para "Detalhes", "Editar" (usando RouterLink) e "Excluir".
  + Emite um evento (@Output() excluir) com o ID da receita quando o botão de exclusão é clicado, para que o componente pai (ReceitaListaComponent) possa tratar a exclusão.
  + Constrói a URL da imagem concatenando imageBaseUrl (propriedade do componente) com receita.imagem\_path.
* **ReceitaDetalheComponent (receitas/receita-detalhe/receita-detalhe.component.ts)**:
  + Exibe todas as informações de uma receita específica.
  + Obtém o id da receita dos parâmetros da rota usando ActivatedRoute.
  + Utiliza o ReceitaService para buscar os dados da receita específica.
  + Constrói a URL da imagem de forma similar ao ReceitaCardComponent.
* **ReceitaFormComponent (receitas/receita-form/receita-form.component.ts)**:
  + Gerencia o formulário de criação e edição de receitas.
  + Utiliza ReactiveFormsModule do Angular (FormBuilder, FormGroup, Validators) para criar e validar o formulário.
  + No modo de edição, carrega os dados da receita existente (obtidos via ActivatedRoute e ReceitaService).
  + Manipula a seleção de arquivos de imagem (<input type="file">).
    - O arquivo selecionado é armazenado em uma propriedade do componente (selectedFile: File | null).
  + No onSubmit, cria um objeto FormData para enviar os dados do formulário e o arquivo da imagem para o ReceitaService. Isso é necessário para uploads de arquivo via HTTP.

**3.3. Serviços**

* **ReceitaService (receita.service.ts)**:
  + Centraliza toda a lógica de comunicação com a API backend PHP.
  + Injeta HttpClient para realizar requisições HTTP.
  + A apiUrl (ex: http://localhost/memoteca\_api) define o endpoint base da API.
  + **Métodos Principais:**
    - getReceitas(): Envia requisição GET para listar\_receitas.php.
    - getReceita(id: number): Envia requisição GET para obter\_receita.php.
    - criarReceita(formData: FormData): Envia requisição POST com FormData para criar\_receita.php. O HttpClient define automaticamente o Content-Type como multipart/form-data.
    - atualizarReceita(formData: FormData): Envia requisição POST com FormData para atualizar\_receita.php. O ID da receita é incluído no FormData.
    - excluirReceita(id: number): Envia requisição DELETE para excluir\_receita.php.

**3.4. Roteamento (app.routes.ts)**

* Define as rotas da aplicação, mapeando caminhos de URL para componentes específicos.
  + Ex: { path: 'receitas', component: ReceitaListaComponent }
  + Ex: { path: 'receitas/nova', component: ReceitaFormComponent }
  + Ex: { path: 'receitas/editar/:id', component: ReceitaFormComponent } (rota com parâmetro id)
  + Ex: { path: 'receitas/:id', component: ReceitaDetalheComponent } (rota com parâmetro id)
* O provideRouter(routes) é usado em app.config.ts para configurar o roteador.

**3.5. Gerenciamento de Estado e Dados**

* O estado é gerenciado principalmente dentro dos componentes ou compartilhado via serviços (embora neste projeto, ReceitaService seja mais para acesso a dados do que para gerenciamento de estado compartilhado entre componentes não relacionados).
* A interface Receita (receita.ts) garante a tipagem e a estrutura dos dados da receita no frontend.

**3.6. Estilização**

* A estilização é feita primariamente com CSS customizado, escopado para cada componente (arquivos .css específicos de cada componente).

**4. Backend (API PHP)**

A API backend é composta por um conjunto de scripts PHP localizados na pasta memoteca\_api (dentro do htdocs do XAMPP).

**4.1. Estrutura**

* memoteca\_api/:
  + db.php: Script de conexão com o banco de dados e configuração de headers CORS.
  + criar\_receita.php, listar\_receitas.php, obter\_receita.php, atualizar\_receita.php, excluir\_receita.php: Endpoints da API.
  + uploads/: Subpasta para armazenar as imagens enviadas pelos usuários. **Importante:** Esta pasta deve ter permissão de escrita para o servidor web.

**4.2. Scripts Principais**

* **db.php**:
  + Estabelece a conexão com o banco de dados MySQL usando mysqli.
  + Define credenciais do banco (servidor, usuário, senha, nome do banco).
  + Configura headers HTTP para Access-Control-Allow-Origin: \* (permitindo requisições de qualquer origem, ideal para desenvolvimento, mas pode ser restringido em produção), Access-Control-Allow-Methods, e Access-Control-Allow-Headers.
  + Trata requisições OPTIONS (preflight requests) para CORS.
* **criar\_receita.php**:
  + Recebe dados via $\_POST (para campos de texto) e $\_FILES['imagem'] (para o arquivo da imagem), pois o frontend envia FormData.
  + Valida os dados recebidos (campos obrigatórios).
  + **Processamento do Upload da Imagem:**
    - Verifica se um arquivo foi enviado e se não houve erros ($\_FILES['imagem']['error'] == UPLOAD\_ERR\_OK).
    - Define o diretório de upload (uploads/).
    - Realiza validações básicas no arquivo (tipo MIME, tamanho máximo).
    - Gera um nome de arquivo único (ex: usando uniqid()) para evitar colisões e sanitiza o nome original.
    - Move o arquivo da pasta temporária do PHP para a pasta uploads/ usando move\_uploaded\_file().
    - Se o upload for bem-sucedido, o nome do novo arquivo é armazenado para ser salvo no banco de dados.
  + Escapa os dados de texto usando $conn->real\_escape\_string() para prevenir SQL Injection.
  + Executa uma query SQL INSERT para salvar os dados da receita (incluindo o imagem\_path com o nome do arquivo da imagem) no banco.
  + Retorna uma resposta JSON indicando sucesso (com o ID da nova receita e o caminho da imagem) ou falha.
* **listar\_receitas.php**:
  + Executa uma query SQL SELECT \* FROM receitas para buscar todas as receitas.
  + Formata os resultados como um array de objetos.
  + Retorna o array como uma resposta JSON.
* **obter\_receita.php**:
  + Recebe um id via $\_GET['id'].
  + Executa uma query SQL SELECT \* FROM receitas WHERE id = ? para buscar uma receita específica.
  + Retorna os dados da receita como uma resposta JSON ou um erro 404 se não encontrada.
* **atualizar\_receita.php**:
  + Recebe dados via $\_POST (incluindo o id da receita a ser atualizada) e $\_FILES['imagem'] (se uma nova imagem for enviada).
  + Se uma nova imagem for enviada:
    - Busca o imagem\_path da imagem antiga no banco.
    - Processa o upload da nova imagem (similar ao criar\_receita.php).
    - Se o upload da nova imagem for bem-sucedido, exclui o arquivo da imagem antiga da pasta uploads/ usando unlink().
    - O nome do novo arquivo é usado para atualizar o imagem\_path.
  + Executa uma query SQL UPDATE para atualizar os dados da receita no banco. A atualização do imagem\_path é condicional (só ocorre se uma nova imagem for enviada).
  + Retorna uma resposta JSON.
* **excluir\_receita.php**:
  + Recebe um id via $\_GET['id'].
  + Busca o imagem\_path da receita no banco.
  + Executa uma query SQL DELETE FROM receitas WHERE id = ?.
  + Se a exclusão do banco for bem-sucedida e um imagem\_path existir, exclui o arquivo da imagem correspondente da pasta uploads/ usando unlink().
  + Retorna uma resposta JSON.

**4.3. Considerações de Segurança (Básicas Implementadas)**

* **Prevenção de SQL Injection:** Uso de $conn->real\_escape\_string() nos dados de entrada antes de inseri-los em queries SQL. Para maior segurança, o uso de Prepared Statements seria recomendado.
* **Validação de Upload de Arquivos:** Verificações básicas de tipo e tamanho do arquivo no lado do servidor. Nomes de arquivos são sanitizados e renomeados para evitar problemas de segurança e colisões.
* **CORS:** Headers configurados para permitir a comunicação entre o frontend (outra origem/porta) e a API.

**5. Banco de Dados (MySQL)**

* **Nome do Banco:** memoteca\_db
* **Tabela Principal:** receitas
  + **Estrutura da Tabela receitas:**
    - id: INT, AUTO\_INCREMENT, PRIMARY KEY - Identificador único da receita.
    - titulo: VARCHAR(255), NOT NULL - Título da receita.
    - ingredientes: TEXT, NOT NULL - Lista de ingredientes.
    - modo\_preparo: TEXT, NOT NULL - Instruções de preparo.
    - historia: TEXT, NULL - História ou descrição adicional.
    - autor: VARCHAR(100), NULL - Nome do autor/origem da receita.
    - data\_criacao: DATE, NULL - Data associada à receita/memória.
    - imagem\_path: VARCHAR(255), NULL - Armazena **apenas o nome do arquivo** da imagem (ex: uniqid\_nomeoriginal.jpg). A URL completa é construída no frontend.

**6. Fluxos de Dados Principais**

* **Adicionar Nova Receita:**
  1. Usuário preenche o formulário no ReceitaFormComponent e seleciona uma imagem.
  2. Ao submeter, o componente cria um FormData com os dados textuais e o arquivo da imagem.
  3. ReceitaService.criarReceita(formData) é chamado.
  4. O serviço envia uma requisição POST para memoteca\_api/criar\_receita.php.
  5. O PHP valida os dados, processa o upload da imagem para a pasta uploads/, gera um nome único para a imagem, e insere os dados (incluindo o nome do arquivo da imagem) na tabela receitas.
  6. PHP retorna uma resposta JSON de sucesso (com o ID da nova receita).
  7. O serviço recebe a resposta, e o ReceitaFormComponent exibe uma mensagem de sucesso e redireciona o usuário.
* **Exibir Lista de Receitas:**
  1. ReceitaListaComponent é inicializado.
  2. ngOnInit chama carregarReceitas(), que invoca ReceitaService.getReceitas().
  3. O serviço envia uma requisição GET para memoteca\_api/listar\_receitas.php.
  4. PHP consulta o banco, busca todas as receitas e retorna um array JSON.
  5. O serviço recebe o JSON e o repassa para o componente.
  6. ReceitaListaComponent atualiza sua propriedade receitas.
  7. O template do ReceitaListaComponent usa \*ngFor para iterar sobre receitas, passando cada receita para um ReceitaCardComponent.
  8. ReceitaCardComponent exibe os dados, incluindo a imagem (montando a URL com imageBaseUrl + receita.imagem\_path).

**7. Conclusão**

O projeto "Memoteca de Receitas" demonstra a integração de um frontend Angular moderno com um backend PHP tradicional para criar uma aplicação web funcional e interativa. A utilização de componentes standalone no Angular e uma API PHP bem definida para operações CRUD e manipulação de arquivos são os pilares desta aplicação.